**Reducering af omkostninger**

**og forlængelse af levetiden**

**For Novencos produkter.**

Erhvervsakademiet Dania, Skive.

Afsluttende Eksamens Opgave.

Dato for aflevering: 21. december – 2018.

Lavet af: Bent Sunesen Mortensen (bent012d)

Vejleder: Ove Thomsen.

# Opsummering

Jeg vil i denne rapport komme ind på hvilken valg jeg har taget, og beskrive hvorfor jeg har valgt at gøre som jeg gør.

Dokumenteringen af den afsluttende eksamens opgave er resulteret i denne rapport og bilag, samt den udviklede applikation og tilhørende database.

Projektperiode med start d. 22-10-2018 og slut d. 23-12-2018 samt aflevering af rapport kl. 12:00.

# Indholdsfortegnelse

Indhold

[Opsummering 1](#_Toc531765061)

[Indholdsfortegnelse 2](#_Toc531765062)

[Introduktion 4](#_Toc531765063)

[Problemstilling 5](#_Toc531765064)

[Problemformulering 5](#_Toc531765065)

[Inception 6](#_Toc531765066)

[Metode og værktøjer 6](#_Toc531765067)

[Unified Process 6](#_Toc531765068)

[Design Studio Method 6](#_Toc531765069)

[Værktøjer 6](#_Toc531765070)

[Programmeringssprog 7](#_Toc531765071)

[Virksomhedsanalyse 8](#_Toc531765072)

[Novenco 8](#_Toc531765073)

[Domæne model 9](#_Toc531765074)

[As-is model 9](#_Toc531765075)

[To-be model 10](#_Toc531765076)

[Brief Use Case 11](#_Toc531765077)

[Prioritering 11](#_Toc531765078)

[Database modellering 12](#_Toc531765079)

[Entity Relation Diagram (ERD) 12](#_Toc531765080)

[Mapping skema 14](#_Toc531765081)

[Database valg 15](#_Toc531765082)

[Data Definition Language (DDL) 15](#_Toc531765083)

[Elaboration 16](#_Toc531765084)

[Iteration #1 16](#_Toc531765085)

[Fully Dressed Use Cases. 16](#_Toc531765086)

[MockUps 19](#_Toc531765087)

[Sekvens Diagram 22](#_Toc531765088)

[Iterationsresultat 23](#_Toc531765089)

[Iterationsevaluering 23](#_Toc531765090)

[Iteration #2 24](#_Toc531765091)

[Use Case #4 25](#_Toc531765092)

[Use Case #8 25](#_Toc531765093)

[Iterationsresultat 26](#_Toc531765094)

[Iterationsevaluering 26](#_Toc531765095)

[Iteration #3 27](#_Toc531765096)

[Design Studio Method 27](#_Toc531765097)

[Entity Relation Diagram genbesøg. 29](#_Toc531765098)

[Mapping skema genbesøg 30](#_Toc531765099)

[Iterationsresultat 31](#_Toc531765100)

[Iterationsevaluering 32](#_Toc531765101)

[Construction 32](#_Toc531765102)

[Mappe struktur 32](#_Toc531765103)

[Database struktur 33](#_Toc531765104)

[Vindues oversigt 33](#_Toc531765105)

[Class Diagram 34](#_Toc531765106)

[Perspektivering 35](#_Toc531765107)

[Bilag 36](#_Toc531765108)

[Design Studio Method resultat 36](#_Toc531765109)

[Interessenter 37](#_Toc531765110)

[Entity Relation Diagram genbesøg 38](#_Toc531765111)

[Mapping skema genbesøg 39](#_Toc531765112)

[Skrald afsnit 40](#_Toc531765113)

[Afgrænsninger 40](#_Toc531765114)

# Introduktion

Denne rapport er blevet til, i et samarbejde med kaastrup|andersen. Dette er et led i k|a´s plan, om at lave digitale ’Internet of Things løsninger’ til deres kunder. k|a har talt med mange potentielle kunder, her i blandt Novenco, som de gerne vil lave en Proof of Concept applikation for.

Novenco er et firma som udvikler og producere car park ventilatorer til udluftning af skadelige gasarter i parkeringskældre. De arbejder på at koble deres ventilatorer på internettet, ved hjælp af Internet of Things. Novenco vil gerne kunne indhente Statusser (Condition Monitoring) for de enkelte ventilatorer.

Udgangspunktet er at en Novenco ventilator indrapporter, dens egen status, om det så er rystelser (vibrationer), strømforbrug, temperatur mm., til en server. Og derfor vil jeg i denne rapport undersøge, hvordan man kan anvende disse statusser, på en måde så det spiller ind i Novencos visioner for reducering af omkostninger og drift af ventilatorer.

Jeg har lavet denne rapport, som et led i min afsluttende eksamens projekt.

# Problemstilling

kaastrup|andersen (k|a) leverer forretningskritiske løsninger, der sætter deres kunder i stand til at opnå markante fordele i en teknologisk og digitaliseret verden, ved hjælp af projektleder konsulenter.

k|a kan hjælpe kunder med at lave Internet of Things (IoT). Det går essentielt ud på at få kundernes produkter koblet til internettet. k|a har i dag, en god forståelse omkring IoT og alt hvad det indebærer af protokoller, sikkerhed, elektronik mm.

k|a står over for en potentiel ny kunde, der hedder Novenco. Novenco laver ventilatorer til parkeringshuse og de har et ønske om at nedsætte udgifterne for den enkelte ventilator, da drifts- og vedligeholdelse, i produktets levetid, har omkostninger der overstiger ventilatorens indkøbspris med en faktor 20. Samtidigt kan Novencos konkurrenter masseproducerer ventilatorer, i andre lande, og lave en billigere indkøbspris, og derved mister de markedsandele.

Novenco har udtænkt den strategi at reducere omkostninger og forlænge levetiden af deres produkter, så den efterfølgende drift og vedligeholdelse vil blive reduceret. For at kunne dette, skal de have noget måleudstyr på deres produkter, som f.eks. måler temperatur vibrationer, strømforbrug, vindhastighed, CO^2, ved hjælp af IoT. Det vil ligeledes skabe grundlag for at man kan lave predictive maintenance. Dette tiltag skal gerne vinde markedsandele tilbage.

k|a har ikke kendskab til Novenco infrastruktur og vil gerne illustrere, hvorledes opsamlet data, fra en ventilator, kan anvendes internt eller eksternt hos Novenco. Hvad skal der ske, når en ventilator sender en fejlmeddelelse? Og hvad sker der i processen efter en fejlmeddelelse? Hvordan kan man sikre at der er en, en til en, cause and effekt på fejlmeddelelser?

# Problemformulering

Hvordan kan synliggørelsen af opsamlede data, bidrage videre i værdikæden for service-montør, kunde eller tredjepart, således det bidrager til reducering af omkostninger og forlængelse af levetiden, for Novencos produkter?

Hvordan kan jeg udvikle en applikation, som kan notificere servicemontøren om ventilations data, der overstiger fastsatte tolerance?

Hvordan kan jeg opsamle servicemontørens fejlrettelser fra en applikation?

Hvordan kan back end løsningen afrapportere generelle fejlrettelser til virksomheden?

# Inception

Denne fase i projektet har været en analyserende fase, den har været med til at danne grundlag for det fremadrettede arbejde. Jeg har i denne fase arbejdet med Metode valg til projektet, virksomhedsanalyse, domænemodeller As-is og To-be, for at skabe en bedre forståelse af projektets domæne. Jeg har lavet Brief Use Cases for at klarlægge projektets omfang.

Jeg har også gennem, arbejdet med Entity Relation Diagram og Mapping, lavet et bud på hvordan databasen kommer til at se ud. Til databasen er der også lavet DDL og mockup data, til at kunne foretage nogle visuelle test, der bekræfter mit arbejde.

## Metode og værktøjer

### Unified Process

Til udviklingsmetode har jeg valgt at bruge udviklingsværktøjet Unified Process. Det er med til at sikre at ens arbejde er af en vis kvalitet mæssig beskaffenhed. Et aspekt af Unified Process er at man vurderer hvilke risici der kan opstå og ved at arbejde med dem tidligt i processen, kan man få afklaret disse usikkerheder, i projekt.

Agile og Scrum?

Ud fra min viden omkring projektet har jeg valgt ikke at anvende Scrum, af flere grunde. En af de grunde er måske mindre vigtig end andre, men som en-mands gruppe vil det ikke give mening at kører med en udviklingsmetode der anbefale at anvende Scrum i teams på 6 plus minus 3.

En anden grund til ikke at kører Scrum, fra et agilt synspunkt, er projektets omfang, da store projekter gerne har mange ønsker og behov og projektets omfang ikke står helt klart fra start til slut. Jeg ser derfor at have en plan drevet udviklingsmetode, som den rette metode i dette tilfælde, da omfanget af dette projekt ikke er for stort til at overskue.

### Design Studio Method

Design Studio Method er et værktøj, man kan bruge til at generer mange konceptuelle GUI designs. Dette værktøj gør det mulig at indhente designs til at løse enhver problemstilling. Værktøjet virker bedst, når man inddrager repræsentanter fra mange områder som f.eks. salg, udvikling, marketing, osv., ved at gøre dette får man en bred vifte af kompetencer, der alle, har forskellige indgangsvinkler, på en given problemstilling.

### Værktøjer

#### Integrated Development Environment

Jeg vil anvende MS Visual Studio 2017, som mit IDE til udviklingen af applikationen. Grunde til dette er at det er et framework der indeholder og samler mange værktøjer i et, som f.eks. live debugging, kompilering, unittestting mm.

Jeg vil anvende Microsoft SQL Server Management Studio til at arbejde med databasen. Den giver mig gode muligheder for at lave og redigere databasen.

Til hosting af databasen har jeg anvendt en Microsoft SQL server. Det har jeg gjort fordi jeg kan have en udgave af databasen til at køre på min computer, og fordi det er en relationel database som dette projekt gør brug af.

Begge disse IDE ‘er er værktøjer jeg kender godt og har arbejdet med under uddannelsen.

#### Andre værktøjer

|  |  |
| --- | --- |
| Google sheets | Jeg har lavet en overordnet tidsplan for projektets forløb i google sheets. Jeg startede med at lave en vejledende tidsplan med de fire forskellige faser, som Unified Process foreskriver, Inception, Elaboration, Construction og Transition.  Til koordinering af arbejdsopgave har jeg valgt at bruge min tidsplan fra google sheets, som er et spreadsheet. |
| Dagbog | Jeg har valgt at fører en dagbog over projektet, da jeg så kan se tilbage på min proces. Det er på Github at jeg fører min dagbog således at den altid er tilgængelig. Github er også et godt redskab at lave dokumentation på.  I dagbogen vil jeg også fører en slags dagsorden, med Næste dags gøremål og hvilket arbejde der er blevet gennemført, og arbejdet på, den aktuelle dag. |
| Github | Til Versionsstyring af min applikation, har jeg valgt at anvende Github. Jeg har valgt at anvende Github fordi det er online og således tilgængelig alle steder med internet, Jeg kunne også have valgt at anvende Team Foundation Server, men så skulle jeg til at sætte mig ind i hvordan dette værktøj fungere og hvordan det skulle konfigureres. Ved at anvende Github kan jeg oprette et repository hurtigt og smertefrit under nogen særlig form for konfigurering, samtidigt bruger Github et simpelt interface til at persistere sit arbejde, hvilket sparer mig tid, tid som jeg kan bruge på at arbejde i stedet for. |
| Draw.io | Til at lave diagrammer og modeller har jeg anvendt onlineværktøjet draw.io. Det er et gratis værktøj. Det indeholder en masse forskellige objekter, der gør sig gældende, inden for systemudvikling, om man så laver diagrammer, Mock Ups, SD, eller ERD, så kan man altid finde det man skal bruge. |

### Programmeringssprog

#### Applikationen

Jeg har valgt at lave applikationen som et Windows Presentation Foundation (WPF) program. Med det udgangspunkt betyder det at applikationen bliver skrevet i C#. Grunden til at jeg har valgt WPF er på grund af C#, da WPF anvender C#, som code-behind. C# er et sprog jeg kender godt til og er vant til at bruge. En anden grund til at anvende C# er at kaastrup|andersen gerne vil tale ind til virksomheder i Danmark, der er et Microsoft land. På den måde er det med til at hjælpe dem i deres markedsføring af k|a.

Dertil kommer at applikationen også anvender Security Assertion Markup Language (SAML) til at lave GUI´s.

#### Database

Til databasen har jeg brugt Structured Query Language (SQL). SQL er det mest udbredte programmerings- sprog til relationelle databaser. Det er også et sprog jeg kender godt og finde mig godt tilpas med.

## Virksomhedsanalyse

Virksomhedsanalysen er lavet ud fra research på Novencos hjemmeside[[1]](#footnote-1).

### Novenco

Novenco er en del af Schako Group.

Essensen af Novenco

Novenco har fokus på miljøet og det aspekt har de inddraget i deres design og fremstilling af ventilationsudstyr. Ventilatorer er Novencos hovedområde og de lægger vægt på at deres produkter er ressourcevenlige, samt at Novencos engagement sikre lang levetid og en sikker miljømæssig produktionsproces. Novencos ventilatorer er en del af industrielle, kommercielle, offentlige og beboelsesbygninger over hele verden. Produkter og tjenester markedsføres og distribueres gennem Novencos omhyggeligt udvalgte datterselskaber og agenter.

Grønt fokus

Novencos forretning er koncentreret omkring de landbaserede applikationer og markeder. Det er også på dette marked Novenco udvikler og implementere deres højeffektive ventilatorer, der kræver minimale ressourcer og reducere udledningen af skadelige stoffer. Car Parks segmentet er Novencos kerneforretning, som Novenco revolutionerede med deres jet-fans i 1990’erne. Disse jet fans har ført til et stigende antal installationer. Selv i kritiske situationer i tilfælde af brand, forbliver miljøet i fokus, da mængden af energi til ventilatorerne er mindst. Disse Jet fans beskytter både liv og miljøet.

Høj kvalitet

Novenco garanterer produkter og systemer af høj kvalitet, der har høj ydeevne i mange år. Kvalitet er et nøgleord fra start til slut i deres design proces. For at bevarer den høje kvalitet bliver Novencos processer kontinuerligt vurderet, for at opfylde design kravene. Samtidig tildeler Novenco den nødvendige tid og ressource til at sikre at nye designs er korrekt og verificeret.

Innovation

Siden 1947 har Novencos design udvikling og drift af ventilation produkter og systemer givet Novenco stor erfaring. Den indsats Novenco lægger i forskning og udvikling afspejler dette og gør Novenco i stand til at skabe produkter på den teknologiske front med hensyn til ydeevne og holdbarhed. I en verden, der er i konstant forandring, er det Novenco ønske at kunne levere innovative løsninger, der er miljøvenlig og lever op til den næste generation af ventilationsprodukter.

Produktionsmiljø

Novencos produkter bliver fremstillet i Danmark og er i overensstemmelse med Miljø Standarderne i henhold til ISO 9001 og ISO 14001. Novenco minimere belastningen på miljøet og reducere energiforbruget, forbedrer sorteringen af affald, minimerer stålskrot og evaluerer deres produktsortiment og leverandører fra et miljømæssigt synspunkt. Den indsats der kræves for at betjene miljøet og markederne med de rigtige produkter er konstant voksende og det er en udfordring Novenco forpligter sig til hver dag at opretholde. Alle Novencos produkter og systemer er certificeret i henhold til ISO 9001:2015 og ISO 14001:2015.

## Domæne model

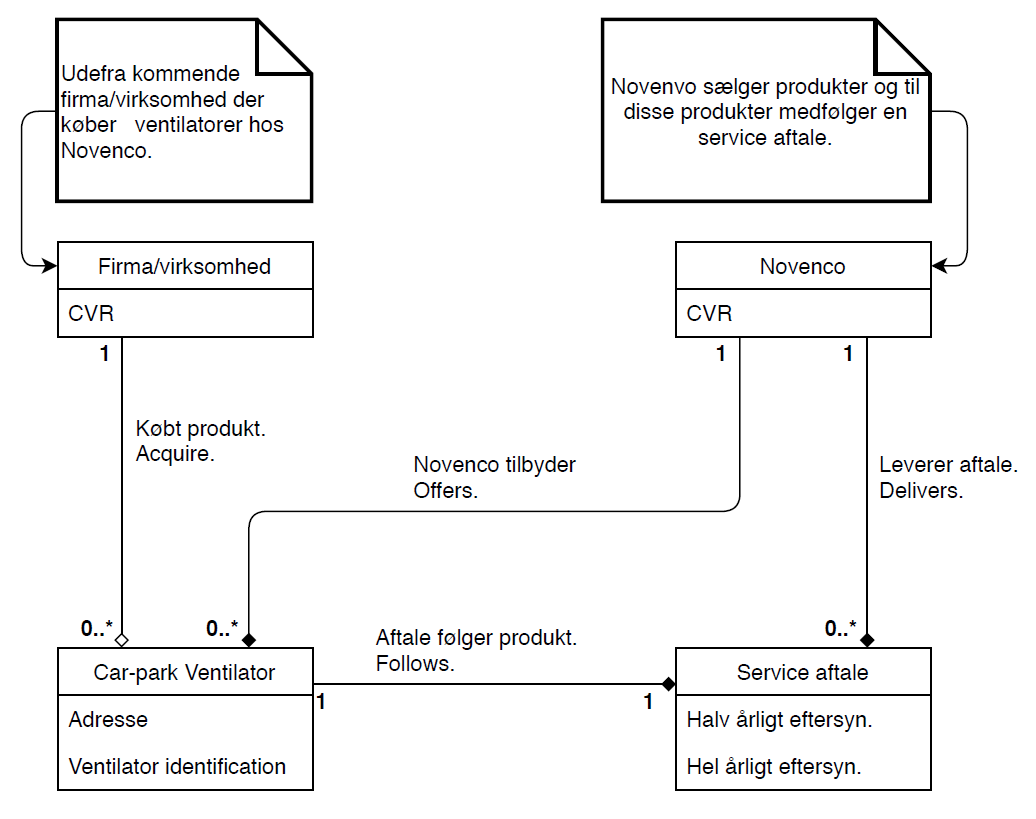
Jeg har I Inception fasen valgt at lave to domæne modeller. jeg vil lave en As-is og en To-be model. Jeg har valgt at lave de to domænemodeller for bedre at kunne forstå relationerne hos Novenco og Novencos kunder, Samt at synliggøre hvorledes de forskellige partnere interagere med hinanden.

Udførelsen af de to modeller, er med til at skabe en dybere forståelse af hvor vi er i dag med As-is. Med To-be vil jeg prøve at vise hvad fremtiden kan tilbyde og gøre anderledes, for dermed at realisere Novencos ønsker.

### As-is model

Min As-is model i *figur 1.* er lavet på baggrund af de oplysninger jeg har indhentet fra kaastrup|andersen og gennem de møder k|a har haft med Novenco, samt informationer jeg har kunnet finde via Novencos hjemmeside[[2]](#footnote-2).

Modellen repræsenterer relationen mellem Novenco og én kunde hos Novenco. Modellen viser også at der foretages en halv eller helårlig eftersyn på car-park ventilatorer, for at sikre drift og venlighold af ventilator. Denne udgift til eftersyn er en fast og kendt udgift, som er sat højt for at sikre kvaliteten af drift og vedligeholdelse.



Figur 1. Domæne model As-is.

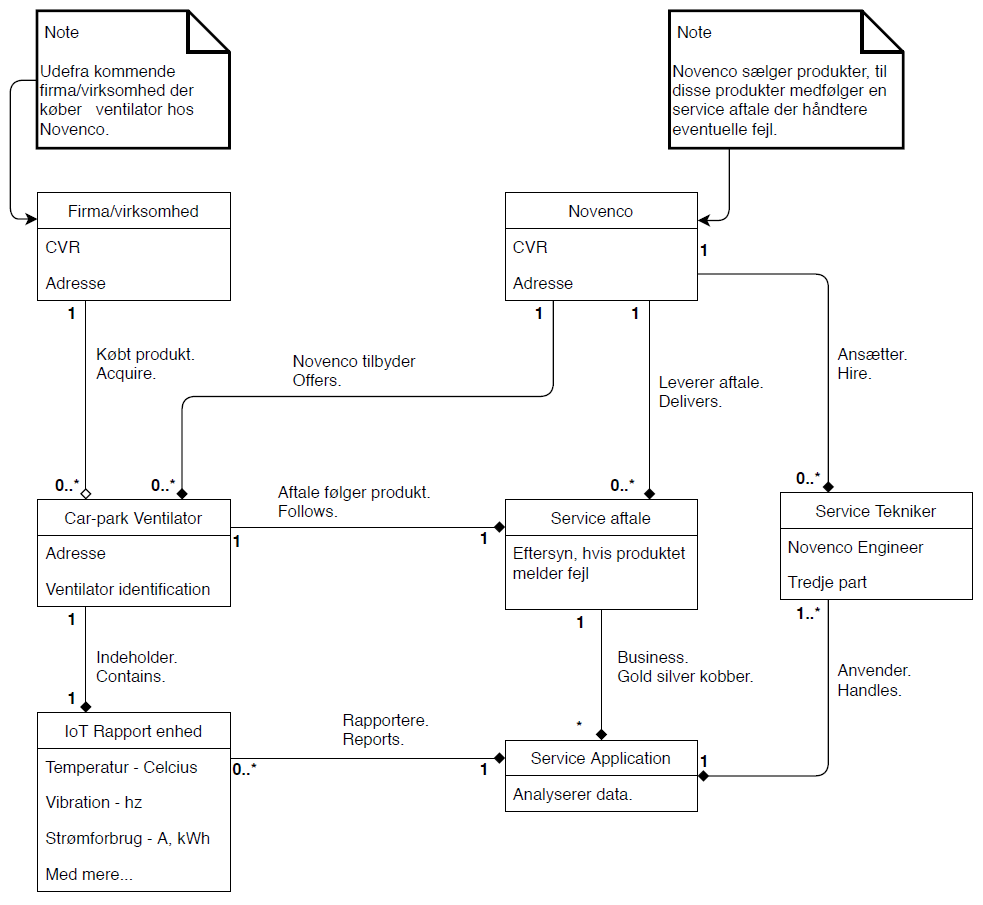
### To-be model

Min To-be model i *figur 2.* er lavet på baggrund af de ønsker, der er fremstillet i problemstillingen. Jeg har også anvendt virksomhedsanalysen i dette skridt.

Denne To-be model er med til at vise, hvordan Novencos domæne kan komme til at se ud. Jeg har tilføjet et ekstra lag med ”IoT Rapport enhed”, ”Service Tekniker” og ”Service applikation”, samtidigt har jeg ændret Service aftalen for at få en forretningsvinkel med ind over projektet.

Det ekstra lag med ”IoT Rapport enhed”, ”Service Tekniker” og ”Service applikation” er med til at kunne eliminere at skulle lave halv eller hel årligt eftersyn, på en ventilator. Hvis der kan nøjes med at lave eftersyn hvert 2. år, fordi ventilatoren først melder fejl efter 2 år, så kan omkostningerne til drift og vedligeholdelse spares væk i den periode.

Serviceaftalen har jeg ændret til at have forskellige serviceniveauer, her kan man forestille sig at en car park har en lav benyttelsesgrad og derfor gerne vil kører med en billigere service aftale, mod at denne car park vil blive nedprioriteret i forhold til en anden car park der har en høj benyttelsesgrad.



Figur 2. Domæne model To-be.

### Brief Use Case

Jeg har valgt at lave Brief Use Cases (*tabel 1.*) for at kunne danne mig et overblik af projektets omfang. Disse Brief Use Cases er lavet på baggrund af mit arbejde med domænemodellen As-is og To-be, problemstilling og problemformulering. Det er disse Brief Use Cases, som jeg vil beskrive dybere i Elaboration fasen som Fully Dressed Use Cases.

I forbindelse med det foregående afsnit kan jeg se at nogle funktionaliteter vil være til gavn for montøren, således han kan se at systemet er aktiv. Således vil montøren kunne se status på normalt kørende ventilatorer i en periode uden fejl, det vil skabe tillid til applikationen.

|  |  |  |
| --- | --- | --- |
| #1 | Modtage fejlmeddelelser på en ventilator. | Brugeren har behov for at kunne modtage fejlmeddelelser på en ventilator, for at kunne rette fejl. |
| #2 | Se en hurtig fejl status for en ventilator. | Brugeren har brug for at se en hurtig fejl status på en ventilator for bedre at kunne danne et overblik over hvilken fejl der er mest kritisk. |
| #3 | Indberette fejlrettelse på en ventilator. | Brugeren skal kunne indberette, hvad årsagen til fejlen skyldes og hvilke tiltag der er blevet taget for at afhjælpe fejlen. |
| #4 | Sætte grænseværdier for en ventilator. | Novenco skal kunne sætte grænseværdier for hvornår en ventilator skal genere en fejl. |
| #5 | Generer mockup data til demonstration. | Jeg har brug for at lave noget mockup data på en ventilator til at kunne teste funktionaliteten af applikationen. |
| #6 | Oprette yderlig fejlrettelse på en ventilator med flere fejl. | Brugeren skal have mulighed for at kunne oprette og indrapportere flere fejl på en ventilator. |
| #7 | Oprette fejl som ikke er meldt ind af en ventilator via IoT. | Brugeren har behov for at kunne indberette ekstra fejl, som er opdaget ved synlig inspektion af en ventilator. |
| #8 | Se normal status på en ventilator. | Brugeren skal kunne se normal status på en ventilator. For at skabe tryghed. |

Tabel 1. Brief Use Cases

### Prioritering

Jeg har valgt at prioritere min liste af Brief Use Cases *Tabel 1.*, så jeg kan arbejde med en del af dem i hver Iteration af Elaboration fasen. Således kan jeg fokusere på enkelte Use Cases i hver iteration af Elaboration faserne. Listen er også prioriteret efter hvilke funktion der er mest kritisk for projektet udvikling. Min prioriterede liste ser således ud *Tabel 2*.

|  |  |  |
| --- | --- | --- |
| Elaboration 1 | Elaboration 2 | Elaboration 3 |
| #1, #2 og #5 | #3, #6 og #7 | #4 og #8 |

Tabel 2. Prioriterede Use Case liste.

## Database modellering

Domænemodellen To-be *figur 2.* har været en stor hjælp i mit arbejde med at lave en database til projektet. Det er ud fra den model og virksomhedsanalysen, at jeg har moddeleret databasen, som den ser ud i dag.

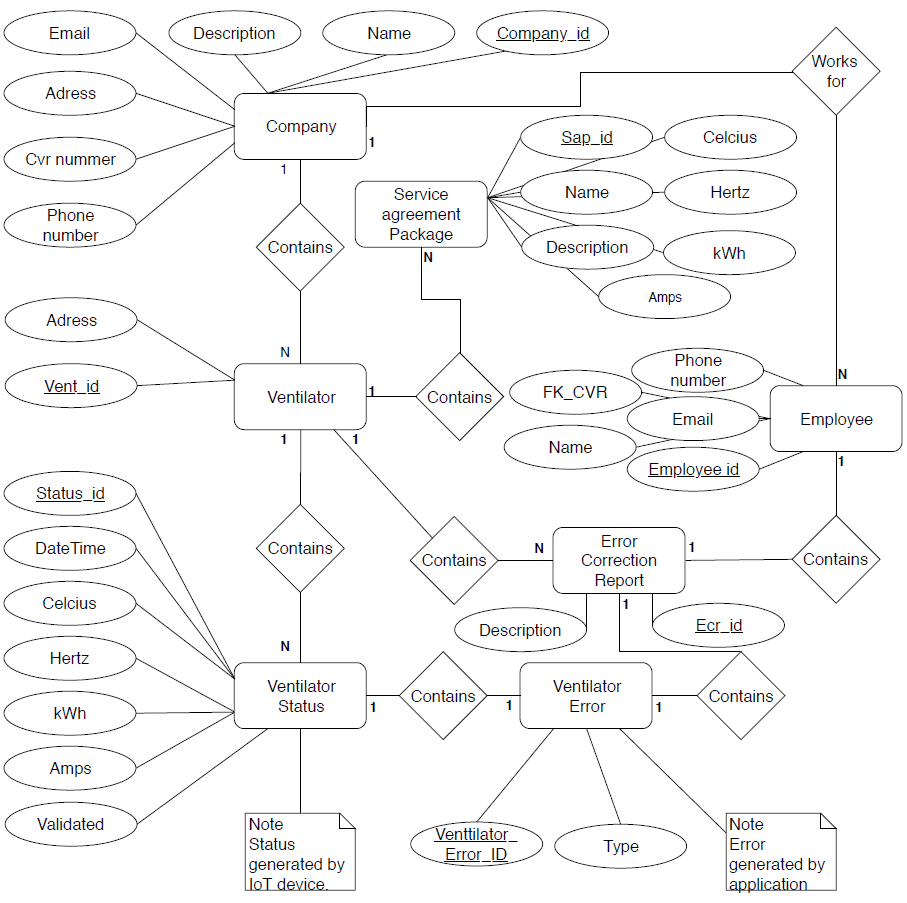
### Entity Relation Diagram (ERD)

Jeg har brugt min domænemodel To-be, som min basis for denne konceptuelle database model se *figur 3.* Ved hjælp af denne har jeg fundet frem til 7 entiteter, som er kernen i denne database, det har været grundlaget for min database opbygning. Essensen af denne database er at registrere fejl og fejlrettelse, i form af en rapport, indrapporteret af en Employee.

De forskellige entiteter jeg har fundet frem til, er Company, Employee, Ventilator, Service Agreement Package, Ventilator Status, Ventilator Error og Error Correction Report. Ingen af mine entiteter er svage da de alle har et unikt id, det sikre at mit system er stærkt og jeg får heller ikke lavet forhindringer for mig selv hvis jeg en dag vil tilføje og lave udvidelser til min database.

Med mit ER diagram vil jeg vise disse relationer.

* Et Company har en Employee, og en Employee har et Company.
* Et Company kan have en Ventilator, og en Ventilator har et Company.
* En Ventilator kan have en Service Agreement Package, men hvis Novenco skal servicere Ventilatoren skal den selvfølgelig have en Service Agreement Package.
* En Ventilator kan have mange Ventilator Statusser, og en Ventilator Status kan kun have en Ventilator.
* En Ventilator kan have mange Error Correction Reports, og en Error Correction Report kan kun have en Ventilator.
* En Ventilator Error har en Ventilator Status, og kan have en Error Correction Report.
* En Error Correction Report har en Employee.

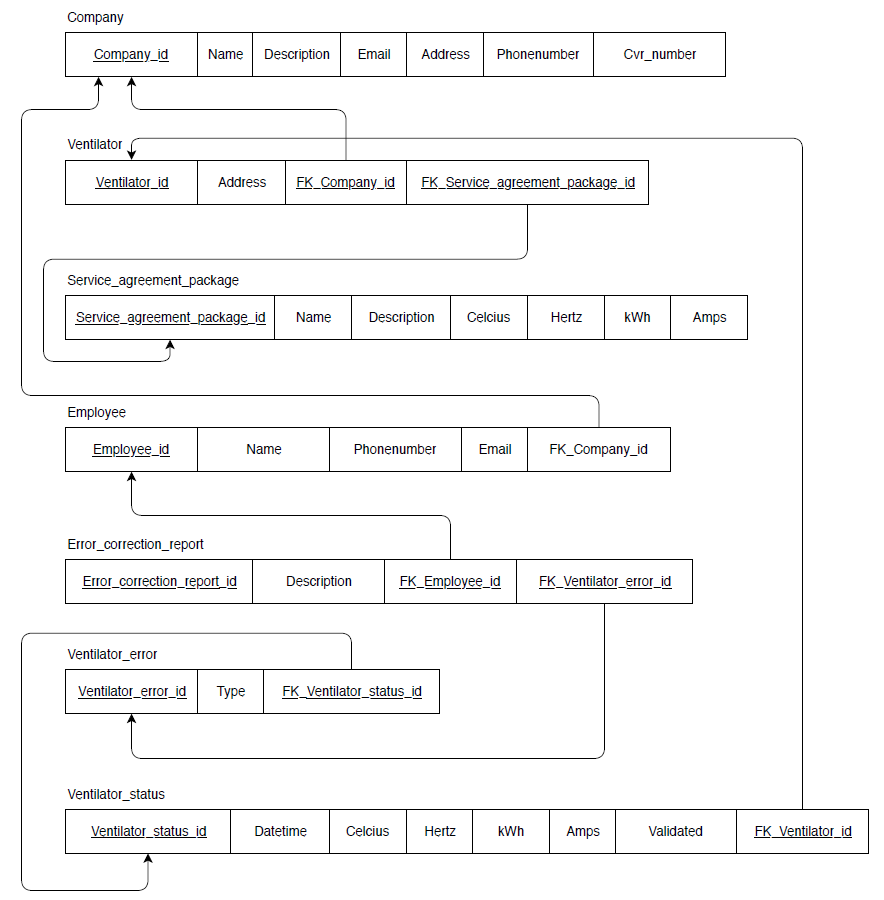


Figur 3. Entity Relation Diagram.

### Mapping skema

Jeg har lavet et mapping skema, se *figur 4.*, til bedre at kunne se relationerne som de vil fremgå i en database, i modsætning til Entity relation diagrammet, hvor relationerne er markeret med streger og symboler mellem entiteterne, er det her i mapping skemaet blevet erstattet med primær-nøgler og fremmed-nøgler. Jeg har valgt at navngive primær-nøgler med tabellens navn efterfulgt af ´\_id´ og fremmed-nøgler starter altid med ´FK\_´. Jeg har valgt at gøre dette i stedet for bare at have et intetsigende navn som ID, nu er det muligt at kunne læse databasen uden at skulle finde dokumentationen frem.

En ting som jeg har ændret fra ERD og til mapping skemaet er navngivningen af tabeller og felter. Dette er sket i forbindelse med at jeg har lavet mapping skemaet og jeg ville gerne have at det er let at genkende og adskille forskellige tabeller på tværs af databasen.



Figur 3. Mapping skema.

### Database valg

Op og indtil dette skridt har processen, med ERD og Mapping, til at strukturere en database været det samme, når man skal lave en database.

Jeg har valgt at anvende en MS SQL Server til at persisterer mit data. Grunden til dette valg er at jeg har godt kendskab til platformen, det er også en relationel database, som passer ind i dette projekt. Samtidig har platformen en kæmpe bruger skare, hvilket betyder at man kan finde mange eksempler og masse af dokumentation på nettet, hvis der skulle opstå problemer med at oprette en database eller lave diverse database kald. Det er et valg der er med til at sikre min arbejdsgang, på en måde, så jeg ikke bliver forhindret i at arbejde videre på grund af problemer.

### Data Definition Language (DDL)

I mit arbejde med ERD og Mapping har det været en relativ hurtig opgave at oprette en database samt lave tables og constraints med DDL. Med lidt online hjælp, har jeg hurtigt kunne finde løsninger på de problemer der opstod, ved at lave DDL for databasen.

Jeg har også genereret noget Mock data til at lægge på min database for at bekræfte at mit arbejde med ERD og Mapping også virker og at det ser ordentligt ud.

# Elaboration

Elaboration fasen er en del af Unified Process, det er her at en stor del af analyse arbejdet til selve applikationen kommer til at foregå. Mange af de ting der er blevet arbejdet og fundet frem til i Inception fasen, vil nu blive uddybet og bearbejdet.

Elaboration fasen er, for dette projekt, blevet delt op i tre iterationer. jeg vil analysere de mest kritiske funktionaliteter i første iteration, ved at få bearbejdet disse funktionaliteter tidligt i processen kan jeg bedre få et billede af om projektet kan blive gennemført med et tilfredsstillende resultat. Nogle Use Cases kan også være afhængig af, at andre Use Cases er blevet lavet færdige, så derfor skal disse også lavet tidligt i processen.

Mit mål med hver iteration er at dokumentere og implementere de prioriterede Brief Use Cases. For at opnå dette har jeg i næste afsnit beskrevet, hvorledes jeg vil håndtere det og hvilke værktøjer jeg vil benytte.

## Iteration #1

I modsætning til Inception fasen, som gik meget på at skulle forstå og finde frem til funktionaliteter af programmet, så vil der i Elaboration fasen også være en del analyse arbejdet.

Jeg vil lave Fully Dressed Use Cases af de prioriterede Brief Use Cases, til denne første Elaboration iteration. Jeg vil Lave System Sekvens Diagrammer for enkelte Fully Dressed Use Cases for at kunne danne mig et billede af hvordan interaktionen mellem program og database ser ud.

I denne fase vil jeg også komme med bud på hvordan programmet kommer til at se ud, ved hjælp af Mock Ups. Disse Mock Ups bliver lavet af de forskellige prioriterede Brief Use Cases det vil gøre arbejdet lettere, når jeg skal til og lavet GUI i WPF.

### Fully Dressed Use Cases.

Jeg har lavet disse Fully Dressed Use Cases for bedre at kunne danne mig et overblik over hvordan applikationen kommer til at se ud rent design mæssig. Jeg har kort noteret i min dagbog, hvilke tanker eller udfordringer der er på dette stadie, det er lidt ligesom hønen og ægget, hvad kommer først, men jeg tror ikke at det gælder om at komme først med noget her, men mere at få det til at spille sammen i en større sammenhæng. Med det sagt betyder det at der har været mange ændringer i Fully Dressed Use Cases og Mock Ups.

Jeg har under denne proces samtidigt tænkt meget over hvordan at programmet bliver bygget op design mæssig og hvordan det kommer til at give den bedste, bruger oplevelse, i anvendelsen af applikationen.

Af de prioriterede Brief Use Cases har jeg lavet 3 Fully Dressed Use Cases i denne iteration

* # 1 - Modtage fejlmeddelelser på en ventilator. *Tabel 2.*
* # 2 - Se en hurtig fejl status for en ventilator. *Tabel 3.*
* # 5 - Generere mockup data til demonstration. *Tabel 4.*

Tabel 3. Fully Dressed Use Case #1.

|  |  |
| --- | --- |
| Use Case name | #1 - Modtage fejlmeddelelser på en ventilator |
| Scope | Novenco |
| Level | Bruger |
| Primary Actor | Bruger |
| Stakeholders & Interests | Bruger som gerne vil se en liste over ventilatorer med fejl |
| Pre-conditions | Brugeren har adgang til internettet. |
| Success guarantee | Brugeren får vist en liste med ventilatorer hvor en fejl er registeret |
| Main success scenario | 1. Brugeren åbner appen. 2. Brugeren vælger sit eget ”navn”, som ”service technician”. 3. Brugeren bliver vist en liste over ventilatorer med fejl. |
| Extension | 2a. Brugerens navn findes ikke.   1. Brugeren anmoder om oprettelse. 2. Brugeren bliver oprettet af systemadministrator. 3. Brugeren vælger sit eget ”navn”, som ”service technician” 4. Brugeren bliver vist en liste over ventilatorer med fejl. |
| Special requirements | - |
| Technology and data variation list | - |
| Frequency of occurrence | Lav |
| Miscellaneous | - |

Tabel 4. Fully Dressed Use Case #2.

|  |  |
| --- | --- |
| Use Case name | #2 - Se en hurtig fejl status for en ventilator. |
| Scope | Novenco |
| Level | Bruger |
| Primary Actor | Bruger |
| Stakeholders & Interests | Bruger som gerne vil se en hurtig fejl status, til vurdering af hvor kritisk fejlen er. |
| Pre-conditions | Brugeren har adgang til internettet. |
| Success guarantee | Brugeren får vist en fejl status for den valgt ventilator |
| Main success scenario | 1. Brugeren trykker på ”Se fejl status”. 2. Systemet åbner et nyt lille vindue, med relevante informationer om fejlen. 3. Brugeren kan åbne yderligere fejl statusser og sammenligne fejl statusser. |
| Extension |  |
| Special requirements | - |
| Technology and data variation list | - |
| Frequency of occurrence | Lav |
| Miscellaneous | - |

Tabel 5. Fully Dressed Use Case #5.

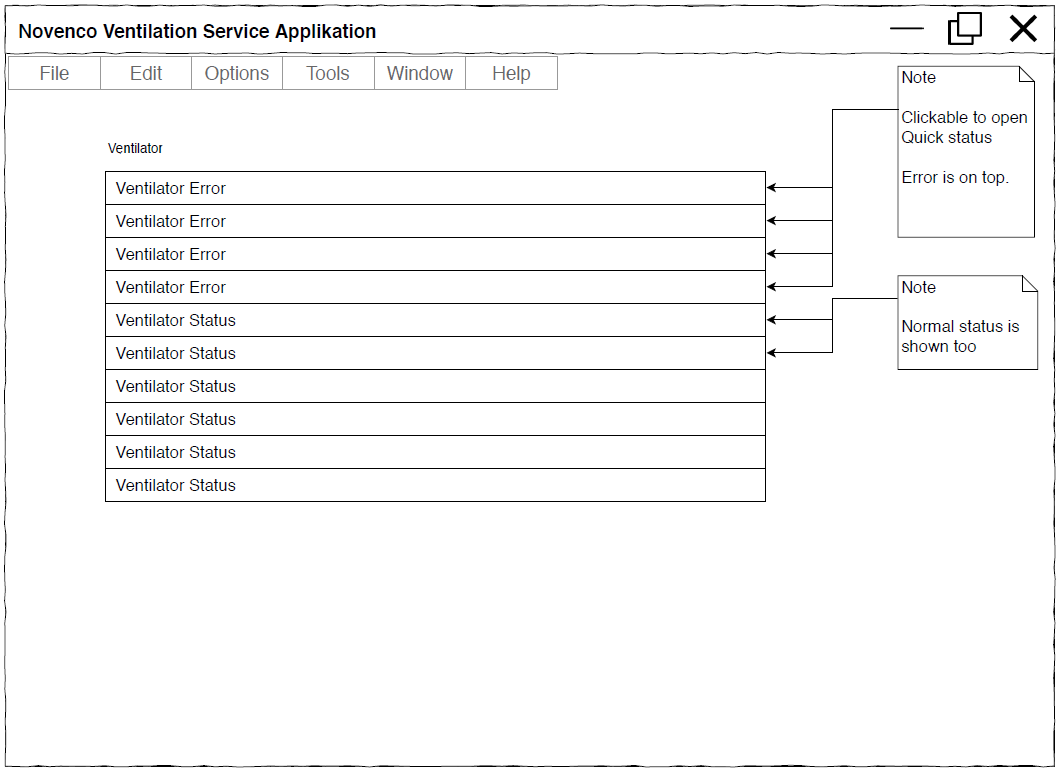
|  |  |
| --- | --- |
| Use Case name | #5 – Generere mock data til demonstration. |
| Scope | Novenco |
| Level | System og Bruger |
| Primary Actor | Bruger |
| Stakeholders & Interests | System |
| Pre-conditions | En fuldt fungerende database. Internet. |
| Success guarantee | Statusser for en ventilator bliver persisteret i en database. |
| Main success scenario | 1. Brugeren afvikler program. 2. Systemet sender nu med et fast interval tilfældigt genererede statusser til databasen. |
| Extension |  |
| Special requirements | - |
| Technology and data variation list | - |
| Frequency of occurrence | Lav |
| Miscellaneous | - |

### MockUps

Til at lave MockUps har jeg anvendt onlineværktøjet Draw.io. Jeg har til inspiration af designet, brugt Fully Dressed Use Cases til applikationen. Det sikre at der er overensstemmelse med Fully Dressed Use Cases og MockUps. Samtidigt gør det også at der er nogle klare retningslinjer for hvad applikationen skal udtrykke og vise med de forskellige GUI interfaces.

#### MockUp Use Case #1

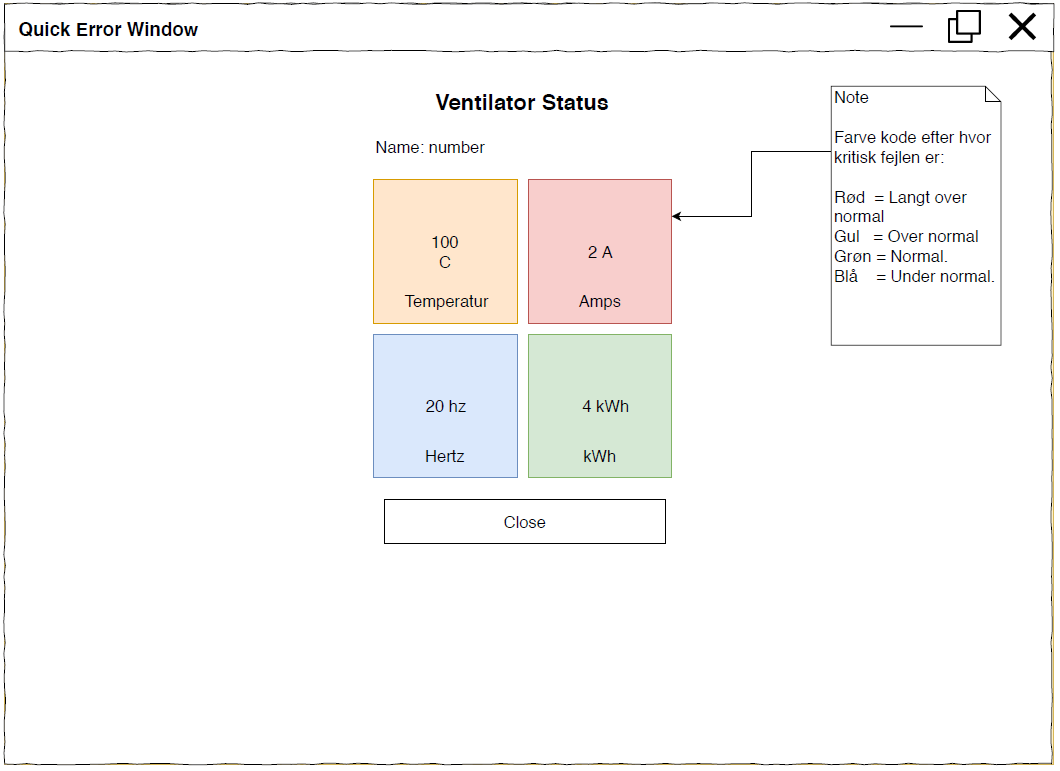
Til Use Case #1 har jeg lavet denne MockUp *figur 4.* Grund ideer har været, at når man starter applikationen op, bliver man præsenteret med dette GUI interface som fremstiller en liste med Ventilatorer. Ventilatorerne bliver listet op, således at dem med en registeret fejl vil figurere øverst på listen med ventilatorer. Hele delen med at modtage fejlmeddelelser, foregår ved at åbne denne applikation. Der vil så ligge en funktion i applikationen som lytter på databasen efter statusser, som kan ligge udenfor den fastsatte grænse.



Figur 4. MockUps. StartSide.

#### MockUp Use Case #2

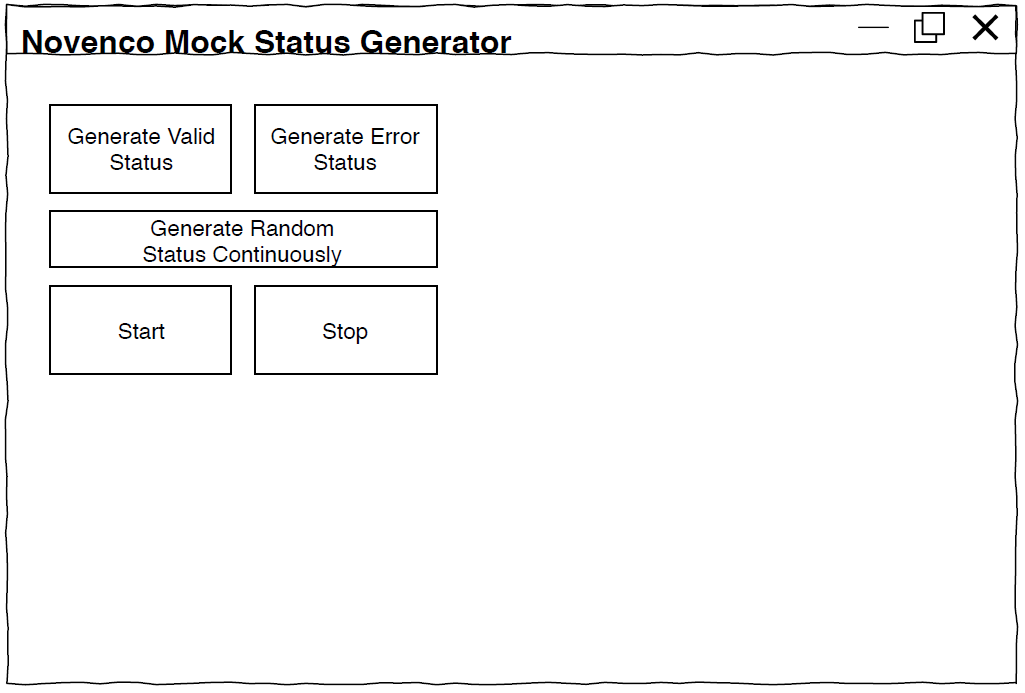
Til Use Case #2 har jeg lavet denne MockUp *figur 5.* Den viser en simpel, men nem at forstå, ventilator status for en ventilator. Servicemontøren kan således åbne flere statusser og holde dem op imod hinanden, her kan han så planlægge sin dag efter afstand mellem ventilatorer eller efter hvilken ventilator der kræver eller har brug for størst opmærksomhed.



Figur 5. MockUp Use Case #2.

#### MockUp Use Case #5

Til Use Case #5 *figur 6.*, har jeg lavet en simpel applikation der skal simulere datastrømmen fra en IoT aktiveret ventilator. Den kommer til at bestå af nogle simple knapper, der kort beskriver hvilken data, ventilator status, der bliver persisteret på databasen. Den er lavet for at kunne vise og teste applikationen.

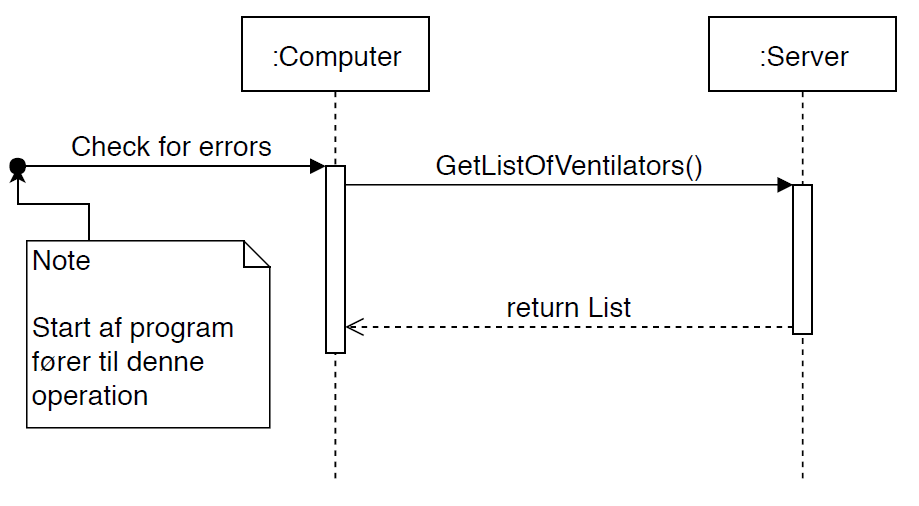


Figur 6. MockUp Use Case #5.

### Sekvens Diagram

Jeg har lavet et sekvensdiagram *Figur 7.*, for at kick starte processen, til at udvikle applikationen. Sekvensdiagrammet viser interaktioner i tidssekvenser,

Et sekvensdiagram viser objektinteraktioner arrangeret i tidssekvens. Den viser de objekter og klasser, der er involveret i scenariet, og rækkefølgen af meddelelser udvekslet mellem de objekter, der er nødvendige for at udføre scenariets funktionalitet. Sekvensdiagrammer er typisk forbundet med brug af case-realisationer i logisk visning af systemet under udvikling. Sekvensdiagrammer kaldes nogle gange hændelsesdiagrammer eller hændelsesscenarier.



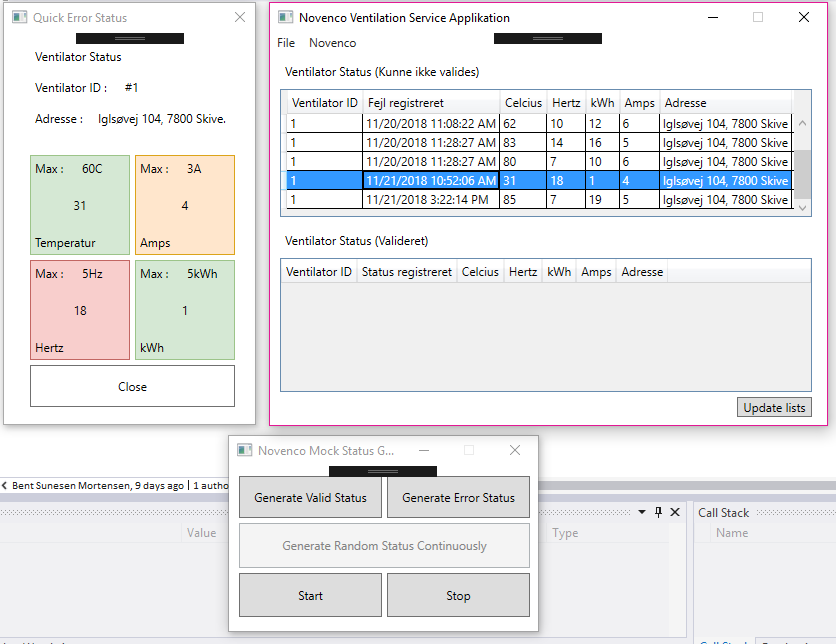
Figur 7. Sekvensdiagram Use Case #1.

Alle udregningerne for om en status, skal genere en fejl eller ikke, foregår på brugerens computer. Her er det vigtigt at applikationen henter data fra serveren omkring grænseværdier og sammenligner disse op mod de målte værdier der kommer fra en ventilator. En ventilator har en service agreement package, der indeholder grænseværdier.

### Iterationsresultat

Jeg vil nu vise resultatet af iteration #1. og beskrive det nærmere.

Applikationen *Figur 8.* er blevet designet ud fra Mock Ups og Fully Dressed Use Cases. Som det fremgår af *Figur 8.* er der sket små ændringer i forhold til de generede Mock Ups. Quick Error Status vinduet har også fået nogle ekstra informationer koblet på for at give mere information til brugeren. Novenco Ventilation Service Applikation vinduet er blevet delt i to lister, hvor en indeholder statusser med registeret fejl og den anden indeholder ventilatorer uden fejl, som derefter forsvinder igen når man opdatere listerne.



Figur 8. Applikation resultat iteration #1.

### Iterationsevaluering

I denne første Iteration har jeg lavet Fully dressed Use Cases, Mock Ups af Use Cases til applikationen og et enkelt sekvensdiagram. Det har været en svær opgave at lave det hele selv og det har da også taget meget længere tid end jeg havde regnet med, en ting jeg synes der er gået godt er rapport skrivning og at huske at føre dagbog over projektet.

Jeg har overskredet min tidsplan med 1 uge og 2 dage. Jeg har brugt for meget tid på at kode applikationen op og har derfor været presset på tiden, her i denne Iteration #1.

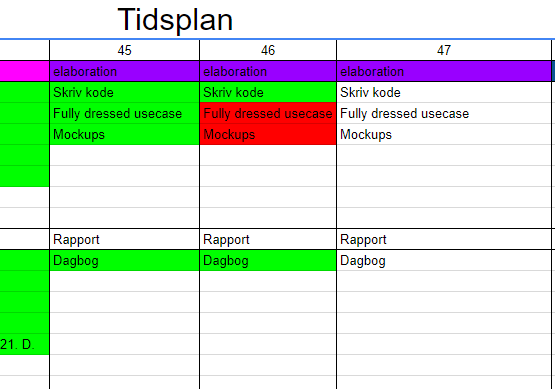
I næste Iteration vil jeg ikke lave Fully dressed Use Cases, Mock Ups eller Sekvensdiagram for at få mere tid til applikationen. Dette har jeg besluttet for at prøve at indhente det tabte. Jeg vil fortsætte med at føre dagbog over projektet.

I næste iteration #2 vil jeg se hvad jeg kan gøre for at rette op på situationen.

## Iteration #2

Tiden er desværre gledet fra mig og det er ikke lykkedes mig at følge min vejledende tidsplan for projektet. Grundende til dette har været en fejl vurdering af tidsestimater for projektet, når man arbejder alene. Ydermere er projektet også blevet forsinket fordi jeg ikke har haft en klar plan for hvordan interfacet skulle se ud og bygges op rent grafisk, forstået på den måde, der har ikke været nogen planer for, hvordan man navigere rundt i applikationen, og det har derfor også spillet ind i, hvorfor projektet er forsinket, på nuværende punkt.

I skrivende stund er jeg midt i uge 47, og som det fremgår af *figur 9.,* skulle jeg gerne være færdig med Iteration #2 og fuld i gang med iteration #3, her i Elaboration fasen.



Figur 9. Udsnit af tidsplan.

Som nævnt i iterationsevalueringen, fra forrige iteration, vil jeg nu se, hvad jeg kan gøre for at indhente min tidsplan igen. jeg har i næste iteration #3. besluttet at bruge værktøjet Design Studio Method i samarbejde med kaastrup|andersen. Dette skal hjælpe mig med det grafiske aspekt og resultatet skulle ende ud i mange konceptuelle Mock Ups til Use Casene #3, #6 og #7. og formodentlig give mig nok feedback til at komme godt fra start.

På grund af møde og andre aktiviteter, i k|a, bliver denne Design Studio Method afholdt torsdag kl. 9:00 i Uge 47, hvor alle indkaldte deltagere har tid. Men det betyder at jeg skal have noget andet på tidsplanen.

På grund af denne forsinkelse og tiltag med Design Studio Method, har jeg valgt at lave om i min prioritering af Use Cases. Jeg kan måske indhente noget af den spildte tid. I iteration #3 skulle jeg arbejde med #4 og #8, Disse to Use Cases er prioriteret mindst i forhold til sværhedsgraden, og jeg vurdere derfor at disse kan laves på mindre tid.

Alt dette betyder at der skal laves nogle rettelser til min prioriterede liste af Use Cases.

Den originale prioriterede Use Case liste ser således ud *Tabel 6*.

|  |  |  |
| --- | --- | --- |
| Elaboration 1 | Elaboration 2 | Elaboration 3 |
| #1, #2 og #5 | #3, #6 og #7 | #4 og #8 |

Tabel 6. Original prioriterede Use Case liste.

Den nye prioriterede Use Case liste kommer derfor til at se således ud *Tabel 7.*

|  |  |  |
| --- | --- | --- |
| Elaboration 1 | Elaboration 2 | Elaboration 3 |
| #1, #2 og #5 | #4 og #8 | #3, #6 og #7 |

Tabel 7. Revideret prioriteret Use Case liste.

Dette betyder dog at jeg har valgt at lægge nogle arbejdsopgaver, der potentielt har en større risiko for at noget kan gå galt senere i processen, hvilket ikke er optimalt.

### Use Case #4

Use Case #4 *Tabel 8.* er rettet mod Novenco og er lavet for at kunne lave en business case til forretningen. Ved at kunne sælge ventilatorer med forskellige Service Agreement Pakker eller rettere sagt service aftaler. Kan de lave en forretning ud af dette. De kan samtidig lave rettelser for hvornår eller hvor tidligt en ventilator skal melde fejl, ud fra disse grænseværdier, der er blevet sat. Jeg har valgt at have tre grundpakker, som udgangspunkt. Disse pakker er Guld Sølv og Kobber, Guld pakken har lavere grænseværdier for, hvornår en ventilator status skal registreres som en fejl, og Sølv er mellem trinnet og kobber er den med de højeste grænseværdier.

|  |  |  |
| --- | --- | --- |
| #4 | Sætte grænseværdier for en ventilator. | Novenco skal kunne sætte grænseværdier for hvornår en ventilator skal genere en fejl. |

Tabel 8. Brief Use Case #4.

### Use Case #8

Til Use Case #8 *Tabel 9.* har jeg hurtig opdaget at jeg kunne genbruge noget af det jeg allerede har programmeret op til Use Case #2. Det har derfor været en hurtig opgave at lave.

|  |  |  |
| --- | --- | --- |
| #8 | Se normal status på en ventilator. | Brugeren skal kunne se normal status på en ventilator. For at skabe tryghed. |

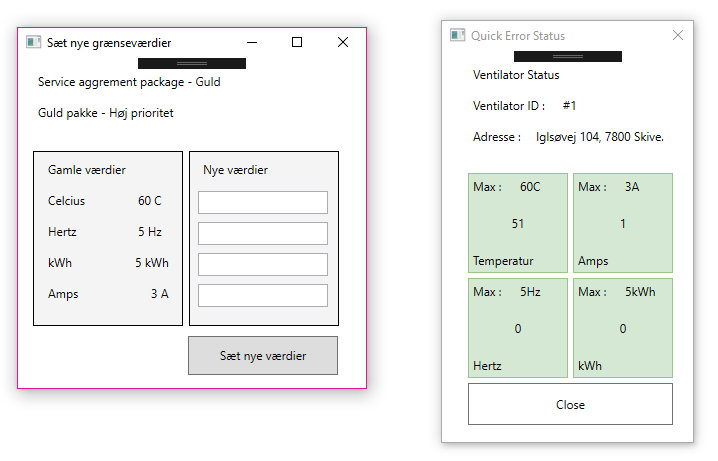
Tabel 9. Brief Use Case #8.

### Iterationsresultat

Jeg vil nu vise resultatet af iteration #2. og beskrive det nærmere.

På billedet til venstre *figur 10.* kan man angive nye grænseværdier for en service pakke. Applikationen vise de gamle værdier, og når man trykker på ”Sæt nye værdier” bliver de gamle værdier i databasen opdateret.

På billedet til højre kan service montøren se normal status for en ventilator, her er det gjort klart med farven grøn at denne ventilator status er iorden.



Figur 10. Applikation resultat iteration #2.

### Iterationsevaluering

Denne iteration er gået helt efter hensigten. Jeg har fået implementeret 2 Use Cases, uden større besværligheder. Det har til dels været på grund af den lave sværhedsgrad af de 2 Use Cases og at der har været mulighed for at kunne genbruge dele af applikationen. Jeg har også fået lagt en plan for at komme bedre i gang med næste iteration. Ved at anvende værktøjet Design Studio Method, kan jeg måske få en bedre start på processen i næste iteration.

## Iteration #3

Jeg vil i denne iteration anvende værktøjet Design Studio Method til at genere konceptuelle Mock Ups til Use Casesene #3, #6 og #7. Jeg har indkaldt interessenter[[3]](#footnote-3), inklusiv mig selv, til møde, i k|a, for at hjælpe med denne opgave.

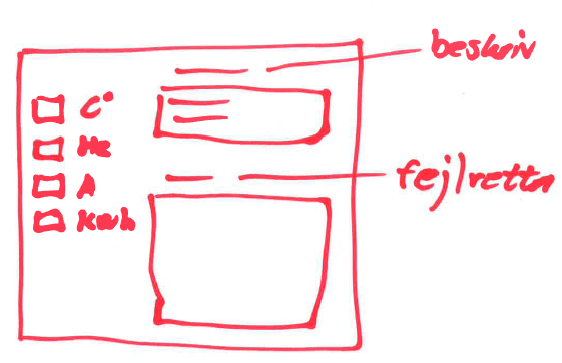
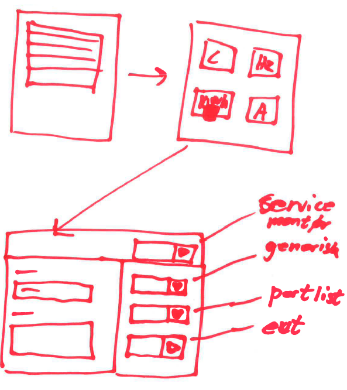
Ved at have byttet om på Iteration #2 og #3, har jeg opdaget at det også er vigtigt at persistere grænseværdierne. Med applikationens nuværende funktioner, kan grænseværdier ændre sig, derfor vil historiske data ikke give mening, hvis disse grænseværdier ændre sig. Jeg har valgt at lægge disse værdier på ”Error\_correction\_report” for at gemme værdierne på det tidspunkt hvor de er aktuelle for en fejl.

### Design Studio Method

Jeg har valgt at bruge Design Studio Method værktøjet, fordi det har været en tidskrævende opgave at skulle designe applikationen selv. Ved at have denne brainstorm øvelse, som Design Studio Method er, har jeg, i samarbejde med interessenter3, på under en time fået genereret 17 forskellige Mock Ups til applikationen.

Under DSM-sessionen fandt vi frem til yderligere funktionaliteter/krav til Use Case #3, f.eks. som at en bruger skal kunne vælge reservedele og at en fejl status skal have en fejlmeddelelse. Disse funktionaliteter kan hjælpe til med at standardisere fejlrettelsesrapporter.

Jeg har her valgt at vise et udsnit af det samlede resultat[[4]](#footnote-4), af Design Studio Method sessionen. *Figur 11.* viser udsnittet med de designs, der komplimenterer applikationen bedst og løser de nye funktionaliteter. Disse designs vil lægge til grund for applikationens udseende fremadrettet.



Figur 11. Design Studio Method udsnit.

#### Fordele og ulemper

Jeg vil her komme ind på nogle fordele og ulemper ved DSM, og hvad min oplevelse af at anvende dette værktøj og hvad jeg har oplevet ved at lave denne øvelse.

##### Fordele

En fordel ved at anvende DSM er tidsaspektet, på én time, er der blevet lavet en masse designs til applikationen, samtidigt er noget af tiden gået med at forklarer processen. Så hvis jeg havde mere tid, end den time der blev afsat til DSM, havde udbyttet også været større.

Vidensdeling er også en fordel, fordi vi var 4 personer, med vidt forskellige baggrunde, kom der også mange forskellige bud, bud som jeg ikke selv havde overvejet at lave.

En anden fordel er at jeg senere kan vise en applikation hvor deltagerne kan genkende dele af deres egen designs i det færdige resultat.

Dialogen, under DSM er helt klart en fordel, det foregår i en positiv tone og alle forslag er velkomne.

##### Ulemper

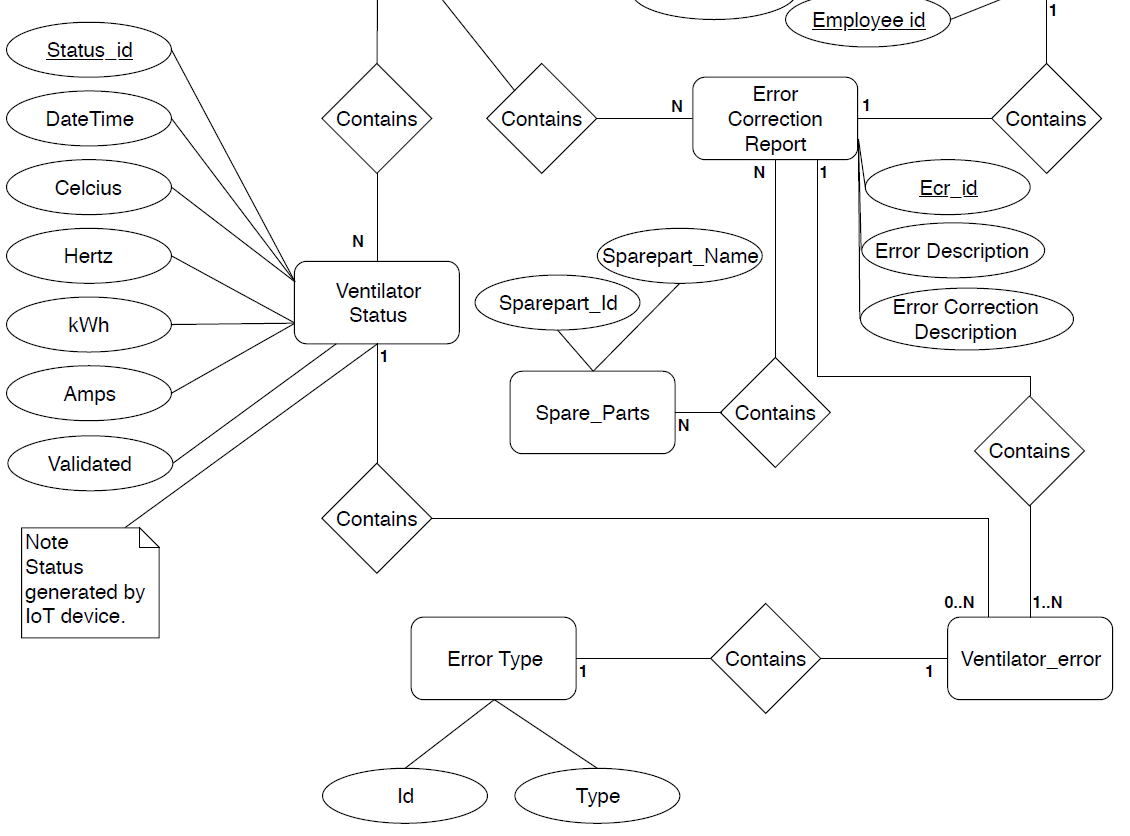
En ulempe, som jeg oplevede ved at lave DSM var at omfangsområdet, blev misforstået, eller også er det ikke lykkedes mig at formidle omfangsområdet i sådan en grad, så deltagerne designer et bestemt vindue (WPF-page). Selvom jeg læste Use Casen op og senere hen viste mit forløb lige arbejde med applikationen, blev designs med applikationens struktur, navigation fra vindue til vindue, ved med at dukke op. Hvis området havde handlet omkring, hvorledes en bruger navigere rundt i applikationen, havde nogen af de designs der blev produceret under DSM, været aktuelle.

Manglende træning. Det var helt klart også en faktor at holdet ikke havde prøvet at lavet DSM før, og det viste sig også, i og med, at en person begyndte at skrive tekst i stedet for at lave designs.

### Entity Relation Diagram genbesøg.

Der er på baggrund af Design Studio Method, blevet fundet nye funktionaliteter. For at imødekomme de nye funktionaliteter, har det været nødvendig at lave ændringer til databasen, da den gamle database ikke imødekom de nye funktionaliteter, derfor vil jeg genbesøge ERD. Jeg vil blandt andet tilføje reservedels lister og en bedre mulighed for at klassificere og generalisere fejl efter type.

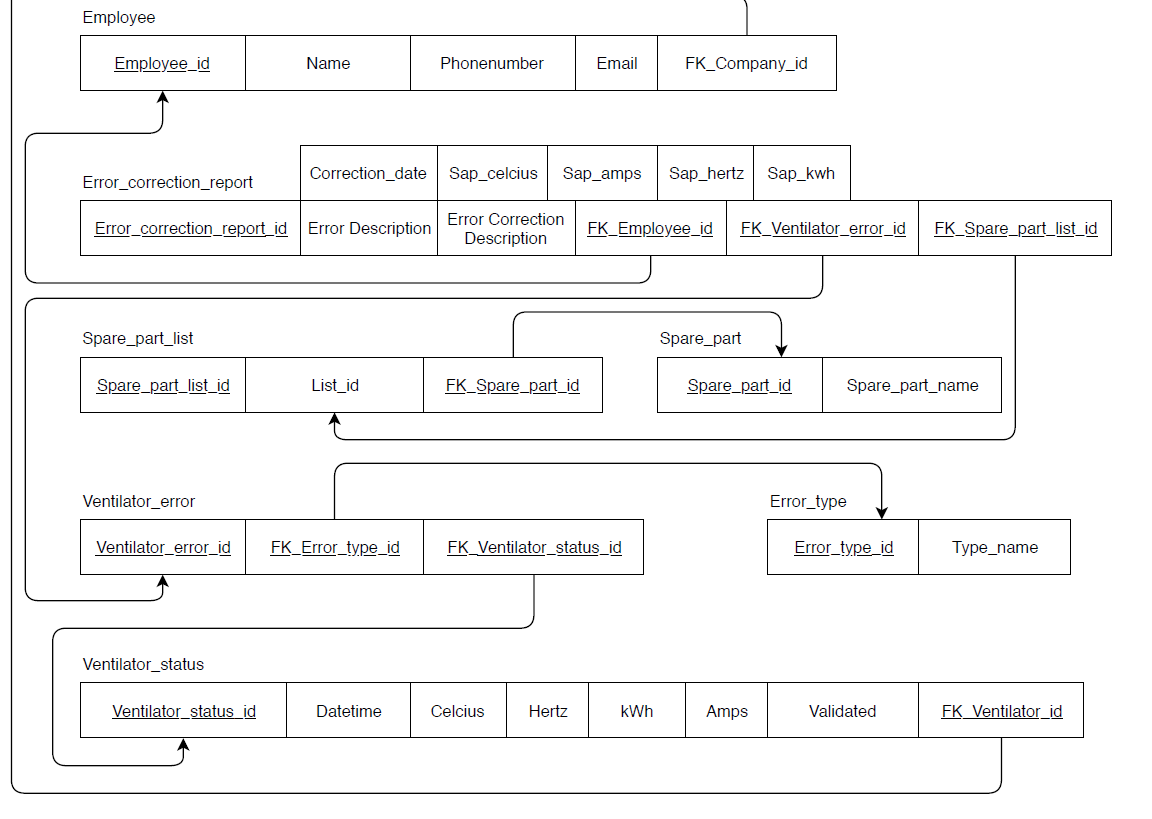
Her på *figur 12.[[5]](#footnote-5)* har jeg taget et udsnit af de ændringer jeg har lavet til ERD for bedre at kunne forstå hvilke ting jeg skal implementere og hvordan det kommer til at ændre strukturen i databasen.



Figur 12. Entity Relation Diagram udsnit.

### Mapping skema genbesøg

Jeg bliver på baggrund af de ændringer jeg har lavet til ERD, også nød til at genbesøge mit Mapping skema igen. Jeg har også her taget et udsnit af mit Mapping skema *figur 13[[6]](#footnote-6).* Under arbejdet for denne gik det op for mig at jeg skal have persisteret de gældende grænseværdier for en fejlrettelse. Disse grænseværdier har jeg valgt at lægge på den eksisterende tabel ”Error\_correction\_report”. Grunden til dette valg, er at jeg kun er interesseret i grænseværdierne på det tidspunkt en fejl bliver rettet. Dertil har jeg også valgt at have en property ”Correction\_date”, på ”Error\_correction\_report”, der indeholder et datostempel for, hvornår en fejl bliver rettet. Det betyder at man nu kan udlede, hvor lang tid der er gået fra fejlen, er blevet registeret og til at den er blevet rettet.

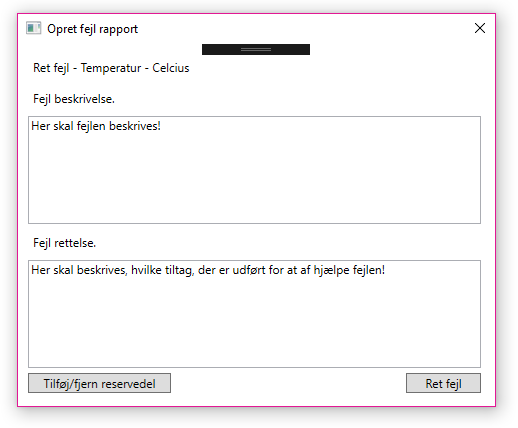


Figur 13. Mapping skema genbesøg.

### Iterationsresultat

Jeg vil nu vise resultatet af iteration #3. og beskrive det nærmere.

*Figur 14.,* viser et vindue hvor der specifikt står udspecificeret, hvilken fejl man er ved at lave en fejl rapport for. Det er et simpelt vindue hvor man skal beskrive fejlen, og komme med bemærkninger omkring fejlen og hvordan den er opstået. Man skal også beskrive hvilke tiltag der er blevet lavet for at rette fejlen. Dertil kan man også tilføje reservedele til en fejl rapport og til sidst kan man trykke på ”Ret fejl”, som så gemmer alle relevante detaljer som dagsdato for rettelsen, reservedelsliste, beskrivelser, gældende service agreement pakke værdier og fejl type i databasen.



Figur 14. Applikation resultat Iteration #3 Use Case #3 og #6

*Figur 15.*, har samme funktion, som *figur 14.,* forskellen her er at på dette vindue bliver man nød til selv at vælge ventilator id samt hvilken fejl type man vil rette. Det er blevet lavet for at kunne generere en ventilator status som hører til en fejl rapport.



Figur 15. Applikation resultat Iteration #3 Use Case 7.

### Iterationsevaluering

Jeg har i denne iteration fået lavet de resterende Use Cases. De 3 Use Cases jeg har prioriteret at lave i denne iteration, har alle noget at gøre med at kunne indrapportere en fejlrettelsesrapport, så ved at indrapportere en fejl ad gangen, bliver Use Case #6 *”Oprette yderlig fejlrettelse på en ventilator med flere fejl.”* og Use Case #3 *”Indberette fejlrettelse på en ventilator.”* opfyldt. Use Case #7 *”Oprette fejl som ikke er meldt ind af en ventilator via IoT.”* bliver også opfyldt, ved at man kan lave en fejlrettelsesrapport uden der er blevet autogenereret en ventilator status.

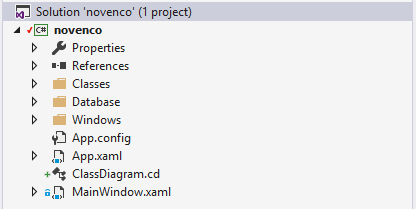
Jeg har i denne iteration anvendt Design Studio Method og jeg synes at det har været en stor hjælp, Jeg har kunnet fokuserer mere på at kode applikationen, i stedet for at sidde og designe applikationen.

## Construction

Jeg vil i denne fase af Unified Process komme ind på nogen af de valg jeg har taget i konstruktionen af applikationen. f.eks. Class diagram fra Visual Studio, vindues oversigt, mappe struktur, Interaktioner og database struktur.

### Mappe struktur

Her på *figur 16.* vil jeg vise hvordan jeg har opdelt mine Classes og WPF-pages (Windows) og Database i forskellige mapper. Jeg har valgt denne fremgangsmåde, for at gøre det overskueligt og ligetil at finde et givent element i applikationen.

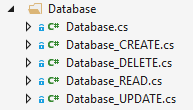


Figur . Mappe struktur.

### Database struktur

På *figur 17.* er der vist, hvordan jeg har valgt at håndtere databaseclassen. Jeg har valgt at inddele databasen i Database\_Create, Database\_Read, Database\_Update og Database\_Delete (CRUD). Jeg har valgt at gøre dette på grund af projektets størrelse og fordi det er begrænset hvor mange databasekald jeg skal have i min applikation. Havde det været et større projekt med mange specialiserede databasekald til forskellige vinduer, ville jeg have valgt en anden løsning.

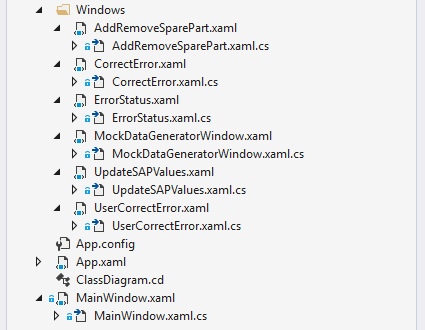
Databaseclassen er en static partial class, hvor ”Database.cs” kun indeholder attributten connection og metoderne OpenConnection(), CloseConnection() og CreateParam(). De øvrige Metoder i databaseclassen er inddelt i de før nævnte databaseclasser. Jeg har valgt at navngive de forskellige databasekald, ud fra den funktion de udfører på databasen, f.eks. som en INSERT statement vil blive navngivet med ”Store” (gem) efterfulgt af ”ErrorCorrectionReport” alt efter hvad denne metode skal persistere i databasen.



Figur . Database struktur.

### Vindues oversigt

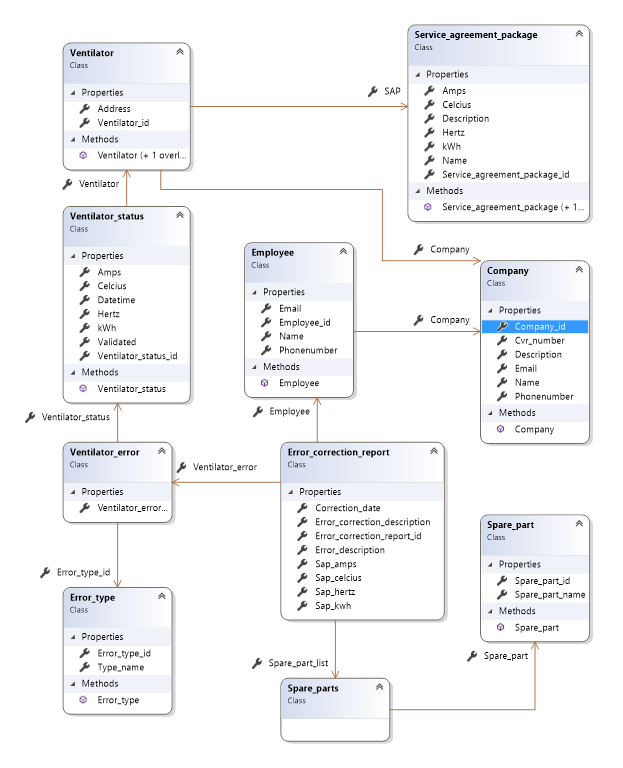
*Figur 18.* viser alle mine vinduer (WPF-pages). Logikken til hvert vindue ligger henholdsvis individuelt i ”\*.xaml.cs” filerne. Jeg har valgt at holde logikken til hvert vindue i dens egen classe, for at skabe et bedre overblik og for at holde det simpelt. Det har også den effekt at jeg ikke deler metoder mellem vinduer.



Figur 18. Vindues oversigt.

### Class Diagram

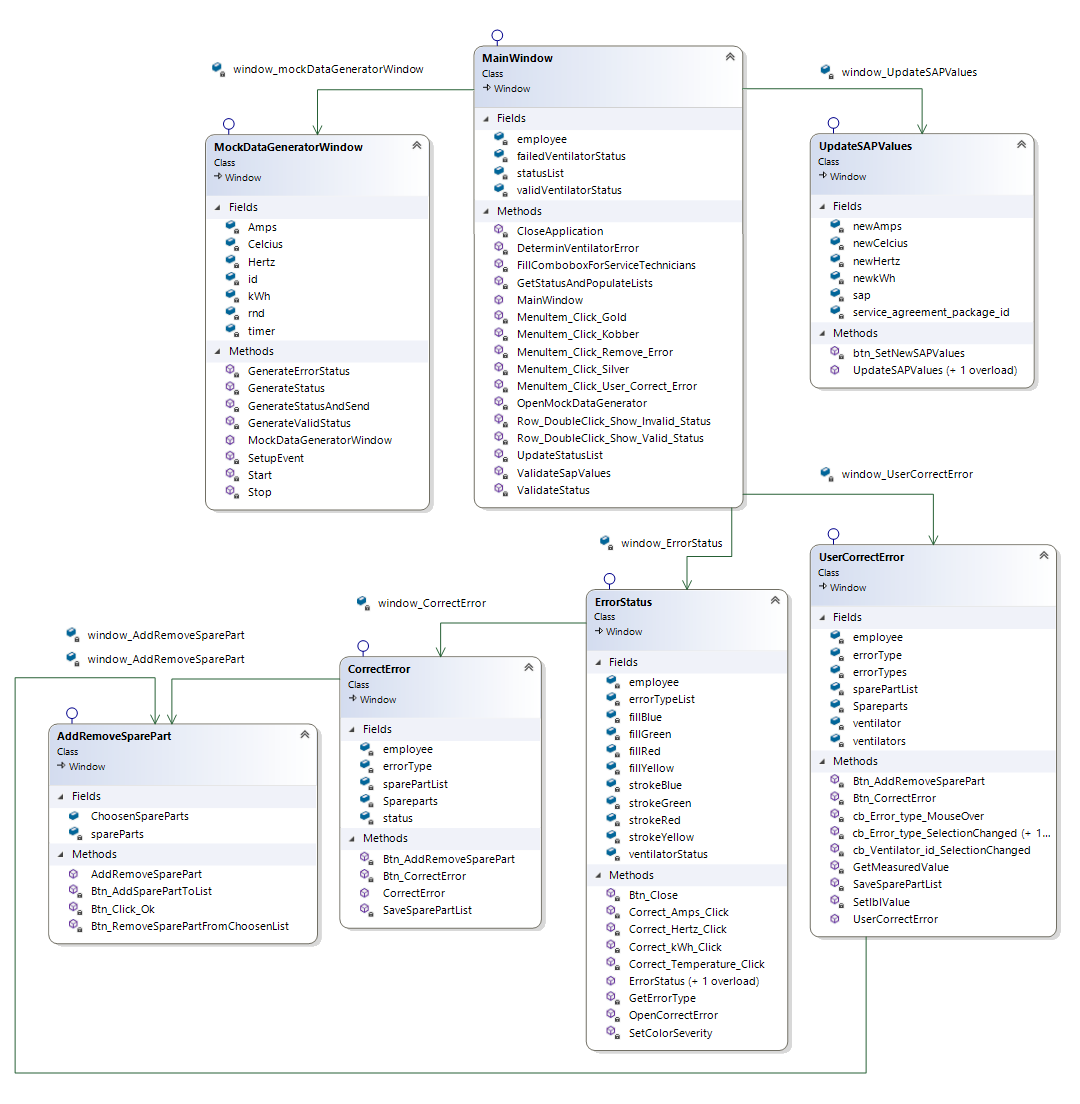
*Figur 20.* viser et billede af mine classer og hvordan de hænger sammen i applikationen. Dette autogenerede Class Diagram, fra Visual Studio, stemmer rigtig godt overens med min konceptuelle model fra To-be og ERD. Den afspejler at det arbejde jeg har lavet, ikke er stukket af og blevet en anden løsning end den jeg har haft som udgangspunkt i starten af projektet.



Figur . Class Diagram.

### Interaktioner

Jeg vil her, på *figur 20.*, vise et Tree-diagram over interaktioner mellem vinduerne i applikationen. På figuren kan man også se hvilke attributter og metoder, hvert vindue gør brug af, i applikationen.



Figur 20. Class diagram - Window Interaktioner.

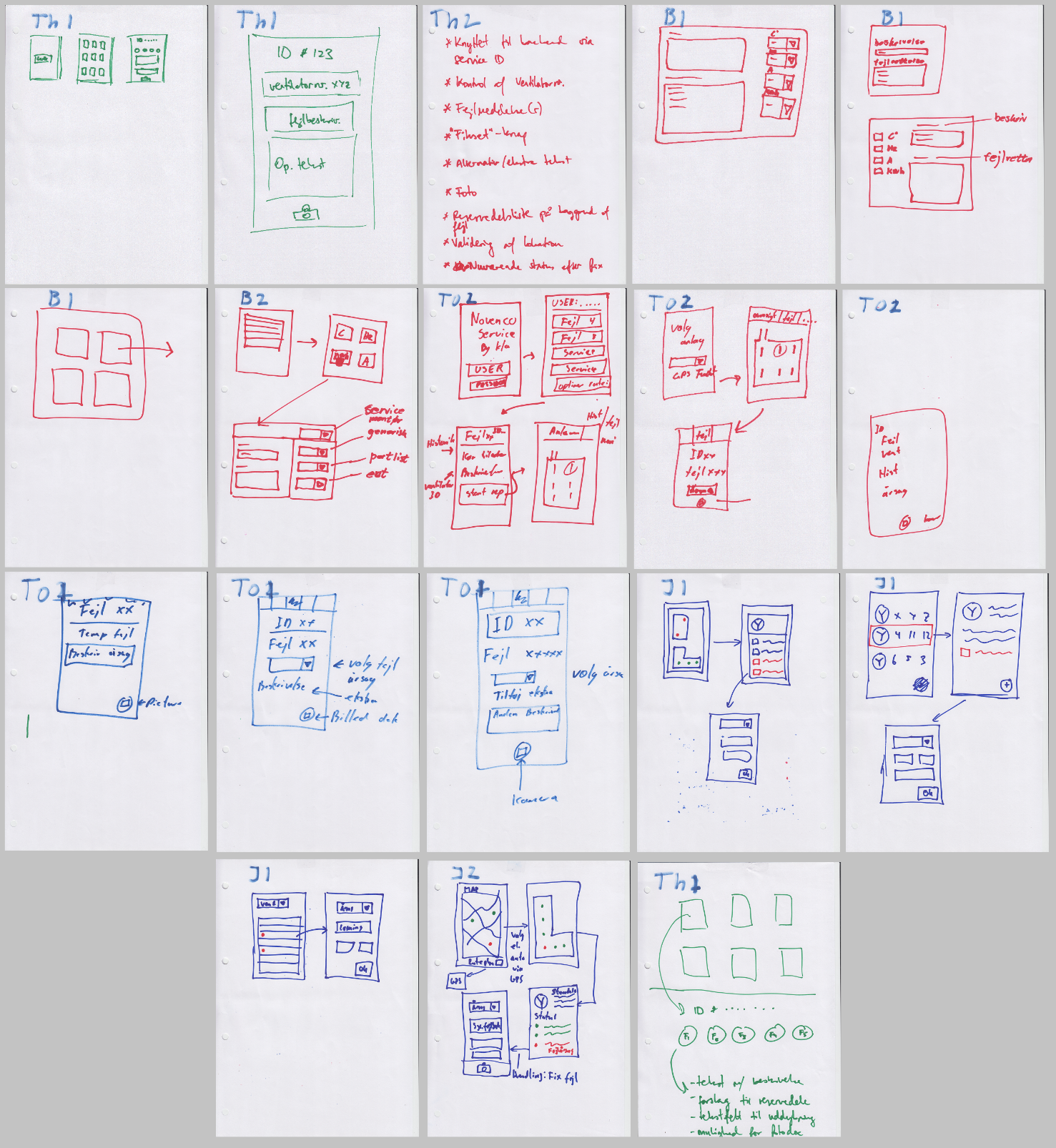
# Perspektivering

Vilde tanker om det efterfølgende perspektiv med applikationen.

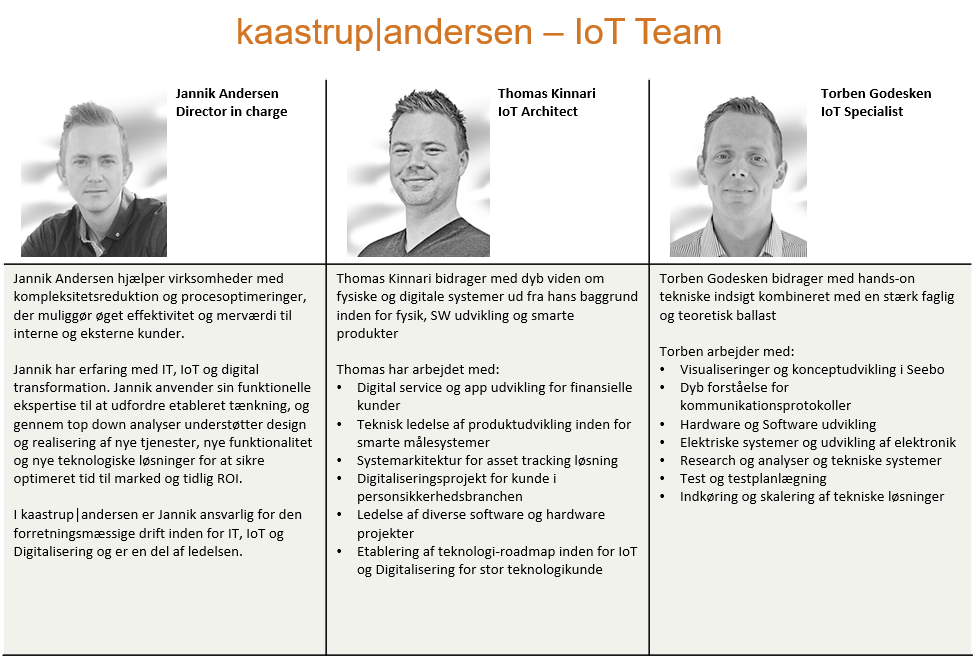
* Man kan kører reservedelslager ind over applikationen.
* Man kan lave flere niveauer på ventilatorer, se skrald afsnit.
* Man kan lave Bi på opsamlede data, se skrald afsnit. Læringen.
* Optimering af materialevalg baseret på historik.
* Service dokumentation, med billede dokumentation.

# Bilag

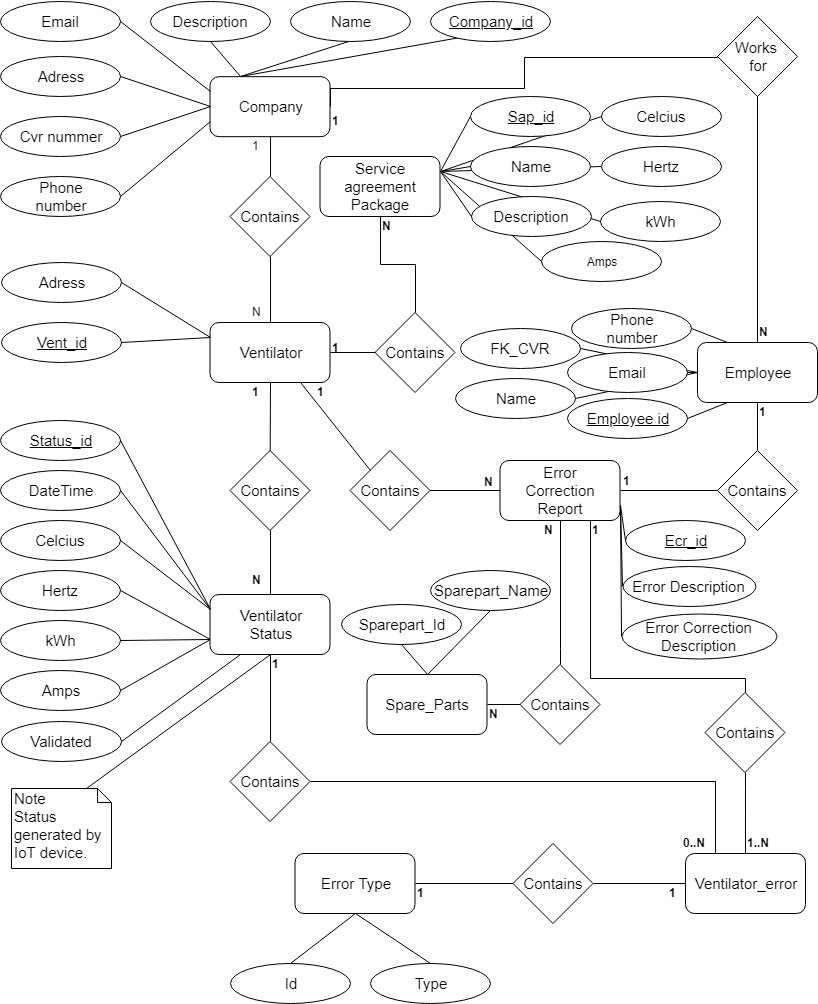
### Design Studio Method resultat



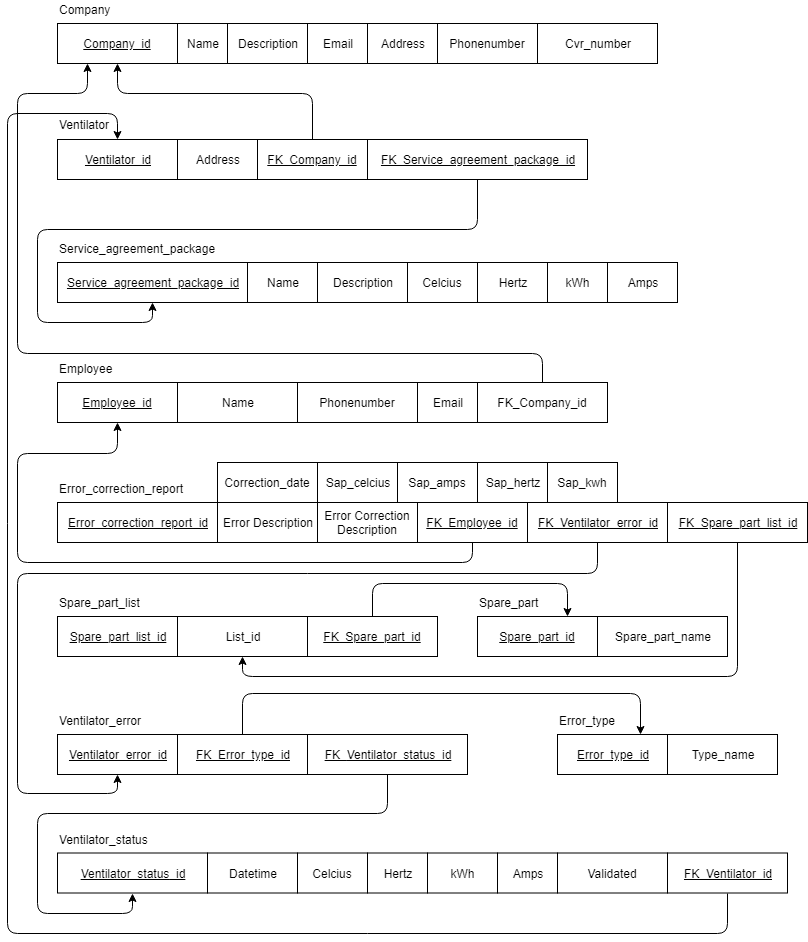
### Interessenter



### Entity Relation Diagram genbesøg



### Mapping skema genbesøg



# Skrald afsnit

### Afgrænsninger

Til dette projekt har jeg lavet nogle afgrænsninger for ikke at udvande det egentlige formål med rapporten. Disse afgrænsninger har jeg ikke kendt fra start af, men det er noget som jeg har opdaget hen ad vejen. Disse afgrænsninger henvender sig mest til databasen.

Jeg har valgt ikke at have mange forskellige produkter(ventilatorer), da det ikke ville give værdi i dette projekt, I en virkelige verden er domænet meget større og der vil selvfølgelig være forskellige produkter, som alle kræver en forskellig tilgang.

Jeg har valgt ikke at lave Car park grupper, for at kunne samle en gruppe af Ventilatorer. Eller for at kunne have en adresse på en gruppe af ventilatorer.

Under introduktion

Jeg vil have nogle afgrænsninger, men de vil stå beskrevet under hvert afsnit, hvis der er foretaget en afgrænsning. Disse afgrænsninger har jeg lavet fordi de, på den ene eller den anden måde, ikke bidrager i nogen særlig grad til problemstillingen/problemformuleringen.

Under Entity relation diagram

Fravalg og overvejelser

Jeg kunne godt have lavet flere niveauer på en ventilator, som f.eks. at have en ´Car park´ entitet mellem Company og Ventilator, for at strukturer det bedre og få en mere virkelighedsnær database, da en ventilator ofte sidder i kombination med andre ventilatorer i en car park. Jeg har fravalgt dette, for at holde det simpelt og således bedre kunne fokusere på problemformuleringen.

Jeg kunne også have trukket adresse i Company og Ventilator ud i en ny entitet, for at spare plads på databasen, og samtidigt overholde normaliseringsreglerne. Jeg har valgt at gøre det på denne måde da jeg igen kun er interesseret i at registrere fejl og fejlrettelser.

Sidegevinst

En sidegevinst af denne database, er at den registrerer statusser for en ventilator, man kan derfor anvende disse data ved hjælp af Business Intelligence. Ved at kører et Business Intelligence projekt på disse data, kan man eventuelt finde, hvilke fejl der vil opstå i den nære fremtid og således kunne lave ´Predictive Maintenance´ på en ventilator inden der sker et nedbrud.

Under fully dressed use cases

Denne proces kommer også til at indbefatte nogle afgrænsninger i forhold til sikkerhed, ikke fordi at sikkerhed ikke er vigtig, men for at fokuserer på problemstillingen/problemformuleringen.

Under domæne model to-be

To-be modellen har samtidigt gjort det klart for mig at jeg skal have nogle afgrænsninger. Afgrænsninger som “worst case scenario” for en car-park ventilator, hvor en ventilator ikke starter, men som heller ikke melder fejl.

1. <https://www.novenco-building.com/> dato: 29-10-2018 [↑](#footnote-ref-1)
2. <https://www.novenco-building.com/> dato: 29-10-2018 [↑](#footnote-ref-2)
3. Bilag - Interessenter [↑](#footnote-ref-3)
4. Bilag – Design Studio Method resultat [↑](#footnote-ref-4)
5. Bilag - Entity Relation Diagram genbesøg. [↑](#footnote-ref-5)
6. Bilag - Mapping skema genbesøg. [↑](#footnote-ref-6)